# Crypto-processeur ECC en RNS sur FPGA avec inversion modulaire rapide

Karim Bigou, Arnaud Tisserand

# Crypto-processeur ECC en RNS sur FPGA avec inversion modulaire rapide

Karim Bigou[2,1] et Arnaud Tisserand[3,1]
[1]IRISA, [2]INRIA Centre Rennes - Bretagne Atlantique, [3]CNRS, Univ. Rennes 1
karim.bigou@inria.fr arnaud.tisserand@irisa.fr

## I. Introduction

La *cryptographie sur courbes elliptiques* (ECC) remplace RSA du fait de clés plus petites et de meilleures performances à sécurité équivalente [1]. L'opération principale dans ECC est la *multiplication scalaire*. Elle nécessite un très grand nombre d'opérations sur le *corps fini* utilisé (ici $\mathbb{F}_p$). L'efficacité de l'arithmétique dans ce corps est donc primordiale. De plus, cette opération doit être protégée contre les *attaques par canaux cachés* (SCA pour *side channel attacks*) comme l'analyse de courant ou du rayonnement électromagnétique.

La *représentation modulaire des nombres* (RNS : *residue number system* [2], [3]) permet d'*accélérer* certains calculs [4]. Elle permet aussi de *randomiser* ces calculs comme *contre-mesure* contre certaines SCA [5]. Mais RNS est une représentation *non positionnelle* qui complique certaines opérations comme l'*inversion modulaire* nécessaire pour ECC sur $\mathbb{F}_p$.

Nous développons un *crypto-processeur* RNS pour ECC sur $\mathbb{F}_p$ à la fois très rapide et robuste face à certaines SCA. Ci-dessous, nous décrivons l'utilisation de RNS pour ECC, l'architecture générale de notre crypto-processeur et son implantation sur FPGA. Nous présentons un nouvel algorithme d'inversion modulaire RNS basé sur l'algorithme d'Euclide binaire étendu qui permet une *accélération d'un facteur 6*.

## II. ECC

Voir [1] pour une présentation détaillée. Une courbe elliptique $E$ sur $\mathbb{F}_p$ peut être définie par l'équation $y^2 = x^3+ax+b$ avec $a, b \in \mathbb{F}_p$ et $4a^3+27b^2 \neq 0$. Les points de $E$ forment un groupe abélien en ajoutant un point spécial. La loi de groupe permet de définir l'*addition de points* $P+Q$ avec $P, Q \in E$ et $P \neq \pm Q$. Pour $P+P$, on parle de *doublement de point* noté $[2]P$. Chaque addition ou doublement de points nécessite une séquence d'opérations dans le corps. La multiplication scalaire est définie par $[k]P = P+P+\cdots+P$ avec $k-1$ additions, $P \in E$ et $k$, le scalaire, un grand entier de 160–521 bits.

## III. RNS

On note $|a|_b = a \bmod b$. L'entier $X$ est représenté en RNS par $\overrightarrow{X}$, ses restes modulo un ensemble de petits nombres $\overrightarrow{X} = (|X|_{m_1}, \cdots, |X|_{m_n})$ [2], [3]. Les $n$ *moduli* $m_i$ sont premiers entre eux 2 à 2 et forment la *base* RNS $\mathcal{B} = (m_1, \ldots, m_n)$ et $M = \prod_{i=1}^{n} m_i$. Le théorème chinois des restes (CRT) assure que $X$ est représenté par $\overrightarrow{X}$ pour $X < M$.

Les opérations d'addition/soustraction et de multiplication sont très efficaces en RNS. Pour $\diamond \in \{+, -, \times\}$ on a :

$$\overrightarrow{X} \diamond \overrightarrow{Y} = \left(|x_1 \diamond y_1|_{m_1}, \ldots, |x_n \diamond y_n|_{m_n}\right) = \overrightarrow{|X \diamond Y|_M}.$$

Les calculs s'effectuent de façon indépendante sur les différents *moduli* sans aucune propagation de retenue entre eux. Ils peuvent être faits en parallèle et dans un ordre quelconque (en particulier aléatoire). Pour $Y$ premier avec $M$, on a aussi : $\overrightarrow{|Y^{-1}|_M} = (|y_1^{-1}|_{m_1}, \ldots, |y_n^{-1}|_{m_n})$.

En contrepartie, les opérations de comparaison, détection de signe/dépassement, division et réduction modulaire sont bien plus complexes, et coûteuses, en RNS. Pour ECC sur $\mathbb{F}_p$, des réductions modulo le premier $p$ sont nécessaires. Pour cela l'état de l'art utilise la réduction de Montgomery combinée avec une opération coûteuse appelée *extension de base* [6].

## IV. Architecture du crypto-processeur

L'architecture de notre crypto-processeur est présentée en figure 1. Elle s'inspire de [4] où le module `cox` a été modifié pour effectuer la réduction modulo 4 en un cycle via le CRT. De plus, quelques pré-calculs supplémentaires sont stockés.

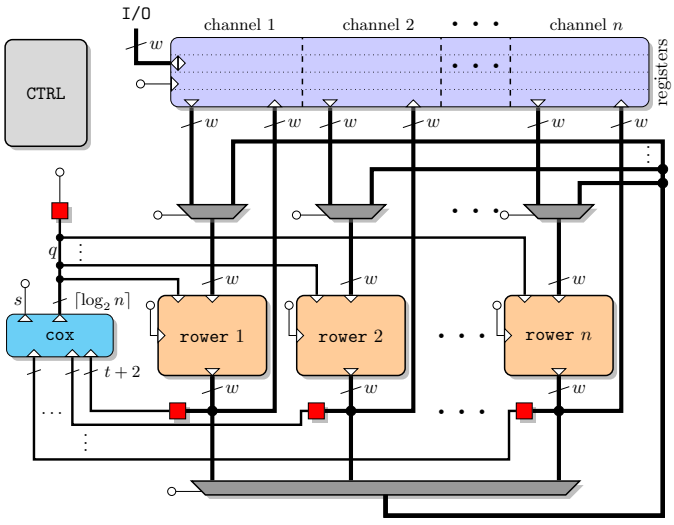


FIGURE 1. Architecture de notre crypto-processeur ECC en RNS.

Les calculs s'effectuent sur les différents *canaux* (1 par modulo $m_i$). Le paramètre $w$ dépend du nombre d'éléments $n$ dans la base RNS et de $p$. La taille des éléments du corps est notée $\ell = \log_2 p$. On a donc $\ell \leq n \times w$. Notre architecture sera évaluée pour différents jeux de paramètres.

## V. Inversion modulaire en RNS

Une inversion modulaire est nécessaire à la fin de $[k]P$. Son coût en RNS représente de 10 à 20 % du temps total. Dans l'état de l'art, les techniques d'inversion modulaire RNS

pour ECC utilisent le petit théorème de Fermat (noté FLT, [4], [7]). Dans la référence d'ECC en RNS [4], l'inversion modulo $p$ se fait avec une exponentiation par $p-2$ qui requiert de nombreuses réductions modulaires. Jusqu'ici, les algorithmes de type Euclidien étaient exclus du fait des comparaisons, divisions et tests de divisibilité trop coûteux en RNS.

Notre solution se base sur l'algorithme d'Euclide binaire étendu [8] et [9, sec. 4.5.2, solution exercice 39]. Pour éviter comparaisons et divisions, nous utilisons l'astuce *plus-minus* de [10] qui ajoute des divisions et réductions par 4. Une nouvelle représentation $\widehat{X}$ optimise certains calculs. Des tests de divisibilité par 4 à faible coût et une modification des autres opérations permettent de conserver la nouvelle représentation dans tout notre algorithme (Algo. 1 noté PMRNS).

**Algorithme 1:** inversion modulaire PMRNS

**Entrées** : $\overrightarrow{A}$, $P > 2$ avec $\gcd(A, P) = 1$
**Sorties** : $\overrightarrow{S} = |\overrightarrow{A^{-1}}|_P$, $S < 2P$

*Initialisation*
**tant que** $\widehat{V_3} \neq \widehat{\pm 1}$ *et* $\widehat{U_3} \neq \widehat{\pm 1}$ **faire**
  **tant que** $|b_{V_3}|_2 = 0$ **faire**
    **si** $b_{V_3} = 0$ **alors** $r \leftarrow 2$ **sinon** $r \leftarrow 1$
    $\widehat{V_3} \leftarrow \text{div2r}(\widehat{V_3}, r)$, $\widehat{V_1} \leftarrow \text{div2r}(\widehat{V_1}, r)$
    $b_{V_3} \leftarrow \text{mod4}(\widehat{V_3})$, $b_{V_1} \leftarrow \text{mod4}(\widehat{V_1})$, $v \leftarrow v + r$
  $\widehat{V_3^*} \leftarrow \widehat{V_3}$ $\widehat{V_1^*} \leftarrow \widehat{V_1}$
  **si** $|b_{V_3} + b_{U_3}|_4 = 0$ **alors**
    $\widehat{V_3} \leftarrow \text{div2r}(\widehat{V_3} + \widehat{U_3}, 2)$
    $\widehat{V_1} \leftarrow \text{div2r}(\widehat{V_1} + \widehat{U_1}, 2)$
    $b_{V_3} \leftarrow \text{mod4}(\widehat{V_3})$, $b_{V_1} \leftarrow \text{mod4}(\widehat{V_1})$
  **sinon**
    $\widehat{V_3} \leftarrow \text{div2r}(\widehat{V_3} - \widehat{U_3}, 2)$, $\widehat{V_1} \leftarrow \text{div2r}(\widehat{V_1} - \widehat{U_1}, 2)$
    $b_{V_3} \leftarrow \text{mod4}(\widehat{V_3})$, $b_{V_1} \leftarrow \text{mod4}(\widehat{V_1})$
  **si** $v > u$ **alors** $\widehat{U_3} \leftarrow \widehat{V_3^*}$, $\widehat{U_1} \leftarrow \widehat{V_1^*}$, $u \leftrightarrow v$
  $v \leftarrow v + 1$
*Corrections finales et sortie*

Le nombre moyen d'itérations de la boucle principale est $0.71 \log_2 p$ (contre $\log_2 p$ pour FLT). De plus, les itérations de PNRMS nécessitent bien moins d'opérations sur $\mathbb{F}_p$. Pour une inversion, on a 6 à 21 fois moins d'additions et 12 à 26 fois moins de multiplications selon le paramètre $n$. Les implantations FPGA (XC5VLX50T pour 192 bits et XC5VLX220 pour 384 bits) montrent des accélérations importantes pour des surfaces similaires (Tab. I/II resp. avec/sans blocs dédiés).

## VI. Conclusion et travaux futurs

Notre avons proposé une inversion modulaire RNS 6 fois plus rapide que l'état de l'art (FLT) pour $\ell = 192$ bits (elle est encore plus rapide pour des tailles supérieures) et ce pour des surfaces équivalentes sur FPGA. Nous travaillons sur d'autres optimisations arithmétiques et de l'architecture, la randomisation des calculs et sur une version ASIC.


## Remerciements

Ce travail a été financé en partie par une bourse de thèse DGA/INRIA et le projet PAVOIS (ANR 12 BS02 002 01).


| Algo. | $\ell$ | $n \times w$ | surface *slices* | surface DSP | surface BRAM | temps $\mu$s |
|---|---|---|---|---|---|---|
| FLT | 192 | $12 \times 17$ | 2473 | 26 | 0 | 72.1 |
| | | $9 \times 22$ | 2426 | 29 | 0 | 60.9 |
| | | $7 \times 29$ | 2430 | 48 | 0 | 90.4 |
| | 384 | $18 \times 22$ | 4782 | 56 | 0 | 193.0 |
| | | $14 \times 29$ | 5554 | 98 | 14 | 258.3 |
| | | $12 \times 33$ | 5236 | 84 | 12 | 242.1 |
| PMRNS | 192 | $12 \times 17$ | 2332 | 26 | 0 | 9.3 |
| | | $9 \times 22$ | 2223 | 29 | 0 | 9.3 |
| | | $7 \times 29$ | 2265 | 48 | 0 | 14.6 |
| | 384 | $18 \times 22$ | 4064 | 56 | 0 | 23.1 |
| | | $14 \times 29$ | 4873 | 98 | 14 | 34.4 |
| | | $12 \times 33$ | 4400 | 84 | 24 | 34.1 |

TABLE I
RÉSULTATS D'IMPLANTATION FPGA AVEC BLOCS DÉDIÉS.

| Algo. | $\ell$ | $n \times w$ | surface *slices* | surface DSP | surface BRAM | temps $\mu$s |
|---|---|---|---|---|---|---|
| FLT | 192 | $12 \times 17$ | 4071 | 4 | 0 | 104.8 |
| | | $9 \times 22$ | 4155 | 4 | 0 | 92.3 |
| | | $7 \times 29$ | 4575 | 0 | 0 | 76.7 |
| | 384 | $18 \times 22$ | 7559 | 0 | 0 | 210.7 |
| | | $14 \times 29$ | 9393 | 0 | 0 | 225.5 |
| | | $12 \times 33$ | 9888 | 0 | 0 | 242.1 |
| PMRNS | 192 | $12 \times 17$ | 3899 | 4 | 0 | 11.6 |
| | | $9 \times 22$ | 3809 | 4 | 0 | 12.0 |
| | | $7 \times 29$ | 4341 | 0 | 0 | 12.4 |
| | 384 | $18 \times 22$ | 7677 | 0 | 0 | 20.9 |
| | | $14 \times 29$ | 9119 | 0 | 0 | 27.7 |
| | | $12 \times 33$ | 9780 | 0 | 0 | 32.5 |

TABLE II
RÉSULTATS D'IMPLANTATION FPGA SANS BLOCS DÉDIÉS.

## Références


[1] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer, 2004.

[2] A. Svoboda and M. Valach, "Operátorové obvody (operator circuits in czech)," *Stroje na Zpracování Informací (Information Processing Machines)*, vol. 3, pp. 247–296, 1955.

[3] H. L. Garner, "The residue number system," *IRE Transactions on Electronic Computers*, vol. EC-8, no. 2, pp. 140–147, Jun. 1959.

[4] N. Guillermin, "A high speed coprocessor for elliptic curve scalar multiplications over $\mathbb{F}_p$," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 6225. Springer, Aug. 2010, pp. 48–64.

[5] J.-C. Bajard, L. Imbert, P.-Y. Liardet, and Y. Teglia, "Leak resistant arithmetic," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 3156. Springer, 2004, pp. 62–75.

[6] N. S. Szabo and R. I. Tanaka, *Residue arithmetic and its applications to computer technology*. McGraw-Hill, 1967.

[7] R. C. C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. X. Yao, "FPGA implementation of pairings using residue number system and lazy reduction," in *Proc. Cryptographic Hardware and Embedded Systems (CHES)*, ser. LNCS, vol. 6917. Nara, Japan : Springer, Sep. 2011, pp. 421–441.

[8] J. Stein, "Computational problems associated with Racah algebra," *Journal of Computational Physics*, vol. 1, no. 3, pp. 397–405, Feb. 1967.

[9] D. E. Knuth, *Seminumerical Algorithms*, 3rd ed., ser. The Art of Computer Programming. Addison-Wesley, 1997, vol. 2.

[10] R. P. Brent and H. T. Kung, "Systolic VLSI arrays for polynomial GCD computation," *IEEE Transactions on Computers*, vol. C-33, no. 8, pp. 731–736, Aug. 1984.